Automatic Verification of a Remote Code Execution Vulnerability Detection Model Using the SPIN Model Checker

By –
Zannatul Ferdous Anusha
ID: 161-35-1595

A thesis submitted in partial fulfillment of the requirement for the degree of Bachelor of Science in Software Engineering

Department of Software Engineering
Daffodil International University

Spring-2020

APPROVAL

This thesis titled on "**Automatic Verification of a Remote Code Execution Vulnerability Detection Model Using the SPIN Model Checker", submitted by Zannatul Ferdous Anusha (Student ID: 161-35-1595)** to the Department of Software Engineering, Daffodil International University has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of Bachelor of Science in Software Engineering and approval as to its style and contents.

DECLARATION

I hereby declare that I have taken this thesis under the supervision of **Md. Maruf Hassan,** Assistant Professor**, Department of Software Engineering, Daffodil International University.** I also declare that neither whole document nor any part of this thesis have been submitted elsewhere for award of any degree.

Anusha

………………………………………

Student Name: Zannatul Ferdous Anusha
ID: 161-35-1595
Batch: 19th Batch
Department of Software Engineering
Faculty of Science & Information Technology
Daffodil International University

Certified by:

[illegible]

………………………………………

Md. Maruf Hassan
Assistant Professor
Department of Software Engineering
Faculty of Science & Information Technology
Daffodil International University

ACKNOWLEDGEMENT

First of all, I am very grateful to my Almighty Allah for giving me the opportunity to walk through the final year. I have learnt civility, ethics, and so on in the previous year of my academic life. I am grateful to all of my teachers for this. I would like to express my heartfelt gratitude to my supervisor Md. Maruf Hassan, Assistant Professor, DIU for giving me the opportunity and continuous support to do this tremendous thesis on the topic of Automatic Verification of A Remote Code Execution Vulnerability Detection Model Using the SPIN Model Checker which helped me in doing a lot of Research and writing of this thesis. Therefore, I have come to know so many new things for which I am so thankful to him. Besides my supervisor, I acknowledge the authority of Cyber Security Center, DIU (CSC, DIU) who have given me the permission to conduct my examination and for the cooperation and support to execute the study.

TABLE OF CONTENTS

TABLE OF FIGURE

ABSTRACT

Background: Communicating and delivering services to the consumers through web applications are now become very popular because of easily wide range information support and integration of data with other applications. Absolute coding practices during the programming and lack of security awareness are the main cause of various types of vulnerabilities at the application level in the World Wide Web (WWW) system. Cyber Attacks are becoming a critical risk for every digital transformation around the world. One of the biggest weaknesses in this area is remote code execution (RCE). According to the Web Application Security Project (CWE / SANS), RCE is on the list of sensitive critical applications that have been vulnerable since 2016. **Objective:** This paper proposes a model, and verify using SPIN model checker. **Results:** This research also simulates the proposed model with automata called finite state machines and exhibits it as a finite model. **Conclusions:** Various types of case studies and detection models are introduced and compared with other vulnerability detection model. After reviewing literatures, we found insignificant researches conducted on justifying web layer vulnerability detection model using SPIN model.

Keywords: Cyber security; vulnerability of web applications; remote code execution (RCE); Finite State of Machine; Automata; User-agent.

CHAPTER 1

INTRODUCTION

1.1 Background Problem

Internet applications currently play an important role in automating traditional daily activities by updating existing solutions. Worldwide, over 3.88 billion people use the Internet and various Internet applications from service providers because it is easy to use and available anywhere, anytime. For the aforementioned beneficial reasons, most organizations or service providers, such as industry, banks, government, education, medicine and other industries, want to offer their services to interested parties via a web application and other online systems. Companies automate processes and offer their clients services via web applications to achieve higher profits with greater customer satisfaction. A modern web application stores the confidential information of organizations and consumers for the above-mentioned reasons: The risk of using these web applications increases daily due to various cyber-attacks. A vulnerability in Internet applications is a serious vulnerability in the system that can be affected. ISPs use the PHP platform to create web applications that simplify the use of code According to OWASP and SANS, the most common security vulnerabilities are SQLi (Structured Request Language Injection), injection of the Cross Site Scripting (XSS) injection buffer overflow system command and Interrupted Authentication Session Management , disclosure of confidential data, remote code execution (RCE), attachment of local files (LFI) etc. However, the new years of remote code execution were a major threat on the Internet that could use the web server function for script / file files.

1.1.2 Remote Code Execution Attacks

Remote code execution is the ability an attacker has to access someone else's computing device and make changes of that computer, no matter where the device is geographically located. Vulnerabilities can provide an attacker with the ability to execute malicious code and take complete control of an affected system with the privileges of the user running the application. After gaining access to the system, attackers will often attempt to elevate their privileges.

1.2 Purpose of This Thesis

Purpose model with Finite State Automata (FSM).

1.3 Research Questions

With having this background and motivation in mind, I pose the following questions:

- Is our proposed model a finite state machine?

1.4 Research Objectives

- To simulate the proposed model with Finite State Automata (FSM).

1.5 Thesis Organization

In this research, APA referencing system has been used in this document. The paper has been furnished with six chapters which are described below:

Chapter 1: Research background, motivation, problem statement, objectives and scopes are conferred in this chapter.

Chapter 2: This chapter includes discussion of the existing related works and figured out the research gap.

Chapter 3: This chapter has the explanation of the vulnerability and its exploitation techniques.

Chapter 4: This chapter contains the research methodology and approaches as it follows for the research.

Chapter 5: Implementation and evaluation are discussed here.

Chapter 6: The outcome of research and direction of further work is presented here.

CHAPTER 2

LITERATURE REVIEW

This section presents previous works in web application layer vulnerability investigation, detection and prevention techniques which focus on the database server.

2.1 Case Study on Web Layer Vulnerability

There is lots of case study available on web layer vulnerability Begum et al.,(2016) conducted a case study on SQLi and RFI based LFI vulnerability where found out the impact of SQLi and RFI based LFI vulnerability based on their exploitation techniques. The study examined 153 web sites where found 45% were in critical condition. Hassan et al., investigated web applications to find out the reasons of BAC vulnerability for analyzing the significant relation with other domains. The research found 39.09% BAC vulnerable web applications among 330 applications. SAT tools provide the most benefits for detecting SQL Injection and XSS vulnerabilities with low false positive rates. A case study explored the WAF based SQLi vulnerabilities exist in the web applications of Bangladesh. Deepa and Santhi Thilagam presented a novel to summarize the current state of the art for securing web applications from major flaws such as injection and logic flaws. A research illustrated the principle of SQL injection, attacks, types and prevention way with details and example. A detailed examination of an SQL attack case was injected against a web service with the goal of testing in action SQL injection attacks. An interactive learning platform capable of grading DML and DDL statements which can expand the range of acceptable SQL statements and eliminate the limitations of current SQL grading systems. A SQL injection detection system has proposed based on an adaptive deep forest model that can automatically detect and improve the precision of the detection during the training process. A framework for trusted remote execution of code is built which can effectively solve the problems of integrating identity authentication, application authentication, and behavior authentication. A research has proposed that explores the efficacy of various proactive mitigation technologies included in the standard exploit protection products and advanced anti-exploitation tools. Microsoft MS-EMET standalone solution was able to block 74%, Symantec, AVAST, kaspersky,ESET all get 100% success. A research has introduced a live wallpaper application that demonstrates how trust can be used even on non-rooted Android devices to gain access to shells. Alwan & Younis (2017) survey the detection and prevention techniques of SQLIA - AMNESIA, SAFELI,WASP, R-WASP, RT-WASP web SQLi vulnerability detection tools. Lawal et al., (2016) provided a systematic literature overview on SQLIA in order to keep researchers up to date with the types, technique, and equipment of SQL injection attacks. Hassan et al., (2018) studied Broken Authentication and Session Management, its exploitation types and their impact in public and private sectors in Bangladesh. 56% websites of our samples were found vulnerable.

2.2 Remote Code Execution Vulnerability

Sommestad et al., proposed an approach to identify the importance of the factors that influence the success rate of remote arbitrary code execution attacks. The experimented success rate varies between 15 and 67 percent for server-side attacks and between 43 and 67 percent for client-side attacks.

2.3 Web Application Vulnerability Detection Model

Fang et al., 2018 represented a tool named WOVSQLI, which can effectively identify SQLIA by implementing a dataset from several sources to detect SQL injection behaviors using word vector and LSTM. Liu and Wang proposed an algorithm (SECOND- ORDER VULNERABILITY DETECTION) which could detect web second-order security vulnerabilities (SQL, XSS and RCE) through two crawl scans. Ross et al., presented a multi-source data analysis system for increased accuracy to detect SQL injection attacks. Kao et al., proposed a framework of SQLi Investigation Architecture (SIA) to proves its feasibility in fighting against of SQLi attacks. V.Nadar et al., developed a model that can detect cross-site request forgery attack and broken authentication and session management attack within the same simulation environment with updated rule libraries and effective test environment. Saoudi Lalia and Ammiche Sarah developed a server side XSS detection approach based on scripts features analysis, which permits detection of a wide range injected scripts without any modification of application source code. Pan and Mao proposed a model for detecting DOM-based XSS using browser extension Grease monkey. Jain et al. proposed a framework to discover web application vulnerabilities by scanning and also mitigating the discovered web vulnerability by customizing configuration rules using Web Application firewall (ModSecurity). Rexha et al. presented an approach to correlate different factors that influence the awareness rising regarding the importance of using secure techniques and paradigms. D. Yadav et al. [16] provided a comprehensive knowledge of web application security and vulnerabilities. CODDLE technique is invent to detect code injection attack by transforming the original data into an encode pattern using Convolutional (Deep) Neural Network [17]. IRONSIDES is developed for implementation of DNS that is known to be invulnerable when running remote code and refusing service attacks on a single package [25]. Alsmadi & Mira (2018) developed a model that test invalid inputs. the research tested the model through evaluating several websites selected randomly. Deepa et al., (2018) Proposed a prototype referred to as DetLogic is developed for detecting different kinds of logic vulnerabilities such as parameter manipulation, access control, and workflow bypass vulnerabilities. Hassan et al., (2018) Proposed and developed a model for detecting LFI vulnerability. The experimental evaluation shown 88% accuracy. Asish Kumar Dalai [28] described the SQL injection attacks and the steps to exploit the attacks and also developed a method to prevent SQL injection attacks.

In view of the above, it is observed that insignificant researches have been focused on RCE vulnerability and its detection. In this paper, we will propose a model for automated detection of RCE vulnerability. We also construct a finite state machine and verify using SPIN model checker.

CHAPTER 3

EXPLOITATION TECHNIQUE

RCE vulnerability with examples is emphasized in the following subsection.

3.1 SQL injection attack

RCE attacks can be differentiated as –

- $_GET METHOD RCE
- $_POST METHOD RCE

3.1.1 $_GET METHOD RCE

Web application allows randomly crawling web pages through the referred links. During the time of crawling the page returns some URL (Uniform Resource Locator) into the address bar. These URL disclose sensitive parameters containing int or string related input or submit functionalities are like below:

http://localhost/test/rcget.php?name=input_name&submit=Submit.

Here the web page is getting input from a user for against his name and the URL is sending the values through the http $_GET protocols. So, this parameter can be easily manipulated by an attacker at this time if the input function refers to the system () function directly then the attacker's input will be dangerous and caused serious harm to the system as it executes every server commands i.e: wget. Vulnerable Coding Pattern is given below. The codding pattern collected from OWASP.

```
<!DOCTYPE html>
 <html>
            <head>
                      <title> RCE GET Method Testing </title>
            </head>
<body>
            <form action="" method="get" >
                        Input your Name : <input type="text" name ="name" /><br>
                      <input type=submit name=submit value=Submit />
           </form>
 </body>
 </html>
<?php if(isset($_GET['submit'])){  $name = $_GET['name']; }
 if(!empty ($name))
{ system("echo $name"); }
 else "input your name"; ?>
```

From the above code we found the system function is allowing all kinds of user inputs without restriction and directly executing to the server. As a result the attacker executes some malicious codes for getting full control on the server.

a. MANUAL PROCEDURE OF CODE EXECUTION

If we consider the following URL:

http://localhost/test/rcget.php?name=input_name&submit=Submit

The input parameter is allowing all inputs. The attacker will then inject the parameter for manipulation like as below:

http://localhost/test/rcget.php?name=input_name; pwd&submit=Submit

This pwd command will show the present directory and all file path's, i.e: name /opt/lampp/htdocs/test. Now the attacker may include and execute their malicious code using wget command.

B. $_POST METHOD RCE

A case of hiding or privacy policy, developers design web application is such a way that the sensitive data will be hidden from the view of general users. Normally designers control this situation using http $_POST protocols. This way they refer all kinds of sensitive data/requests will be send through the $_POST methods.

The example of vulnerable coding pattern.The codding pattern collected from OWASP.

```
<?php
 if(isset($_POST['submit'])){  $name = $_POST['name']; }
if(!empty ($name)){ system("echo $name"); }
else "input your name";
?> <!DOCTYPE html>
```

```
<html>
 <head>
 <title>RCE POST Method Testing </title>
 </head>
<body>
<form action="" method="post" >
 Input your Name : <input type="text" name ="name" /><br>    <input type=submit
name=submit value=Submit />
 </form> </body>
 </html>
```

The above code describes that the user's inputted values are passing through the http $_POST methods and all data are hidden from interacting a general user. But the application is still vulnerable to attack by any attacker as a huge number of tools and process are defined to check the http $_POST and cookie values e.g: butpsuite, http_header, cookie_manager and mostly the Firefox's officially provided add-on hackbar.

a. MANUAL PROCEDURE OF CODE EXECUTION

Here our testing application url is http://localhost/test/rce_post.php, it will show a input box for getting input of application's user's name. After entering the value it doesn't pass the request thorough the url bar directly. In the backend it uses the http methods. I.e: if user searches for "Jack" (name_value), the data "Jack" will be pass through the post and will be found into the $_POST data. E.g: "? name=Jack&submit=Submit".

CHAPTER 4

RESEARCH METHODOLOGY

In this section it is described how our model works to detect RCE vulnerability. Firstly the model is divided into two modules – crawling and RCE detection mechanism. The proposed model drew using draw.io an online based tool.

Working process of this model began with validation of entered URL. The proposed model tests the HTTP 200 web response code to ensure that the web application is working or not. If the server is online, then send URL to crawl; otherwise give a warning and exit.

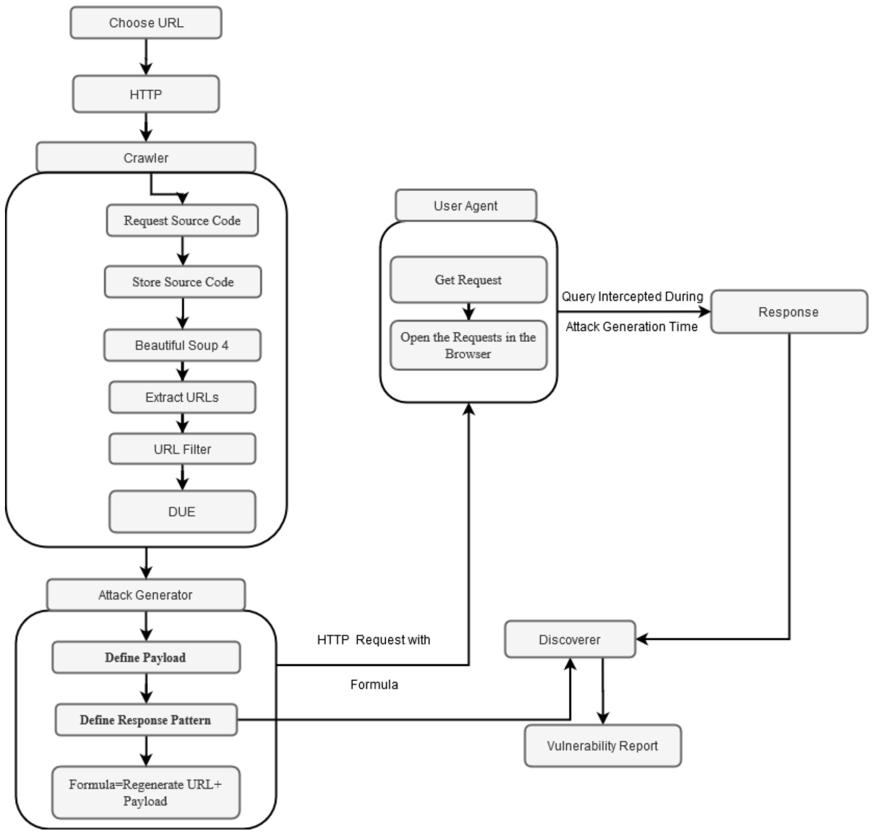


Figure 1: System Architecture of RCE Detection Model

4.1 Crawler

In crawling phase, a request is sent to the target web application for source code using BS4 (Beautiful soup 4) and store in a temporary variable. To find out all possible URLs of the target web application, the specified code will be analyzed. The model must analyze the parameterized URLs which may include RCE vulnerability. The proposed model selects only the parameterized URLs to find out the vulnerabilities.

4.2 HTML/JavaScript Analyzer

To find out the working process of a parameter, we observe parameters passed in a web application through HTTP request and response analysis. In this process every request is waiting for a certain number of parameters where each parameter has a name and value. Several researches are available for the detection of RCE vulnerabilities that are conducted through HTML/JavaScript analyzer [Gupta, M. K. et al.]. This analyzer is used for designing web pages to achieve the restrictions in each form field. The HTML JavaScript parser is designed for parsing a web page in order to make restrictions (if any) in each form field. Client-side validation is done on the browser side using scripting languages such as JavaScript, HTML. At the server code level, an offender can avoid bypassing validation by disabling and therefore disrupting JavaScript business logic. To ensure that the limit is at all, the parameter is inserted on both the client and the server side. JavaScript routines are considered to accept restrictions. The HTML / JavaScript parser is designed for parsing a web page in order to make restrictions (if any) in each form field. Client-side validation is done on the browser side using scripting languages such as JavaScript, HTML. At the server code level, an offender can avoid bypassing validation by disabling and therefore disrupting JavaScript business logic. To ensure that the limit is at all, the parameter is inserted on both the client and the server side. JavaScript routines are considered to accept restrictions.

4.3 Identifying JavaScript validation code

The analyzer must allocate the code fragment which is important for the validation of the parameters and appreciate the working process of these code fragments accumulating the variable temp_variable. This can be complicated because the validation procedure can be performed in two different ways: (1) when a form is submitted and (2) each time the user enters or modifies data on the form using managers of events. JavaScript patterns are systematically performed by the state machine. The state machine consistently imitates the event execution of JavaScript. Each state represents the data entered by the user and the flag that indicates the data that contains an error. When the user submits or modifies the data, the JavaScript code validates the data and updates the error flags accordingly, causing a state transition. The constraints imposed by clients on specific datasets could theoretically depend on the path that the user enters via the state machine, where the temp_variable formula could depend on the structure of this state machine.

4.4 Analyzing validation code

If in the validation procedure the variables temp_variable is identified, they must be analyzed. Such a code can cover several functions, each of which can contain numerous control paths. Each of these control paths can illustrate a unique set of input constraints, which requires the analysis of all the steps during the action. Improved JavaScript can apply a contraction that does not depend on user input, such as sending a form frequently to a global variable. The challenge is to extract only the limitations on entries from a particular part of the JavaScript validation code. BeautifulSoup (BS4) addresses these challenges by implementing a mixed web scraper using RCE Detection mechanism. This application provides scope for all control paths in the verification code and reproduces verification of data used by users.

4.5 Resolving RCE Vulnerability

BS4 extracts and stores all possible URLs from source code in a variable named temp_variable. The existence of parameterised URLs is identified from the stored source code. If any parameterized URL is found then ADT- RCE selects and splits the parameterized URL to generate possible vulnerable links. It then compares the created script to predefined script and returns vulnerable links with potential SQL injection if the script matches.

4.6 Attack Code Generator

Logical formulas passed to the Attack code generator are written in the language of string constraints. These components work with individual tasks, generating new logical formulas whose solutions correspond to hostile data. The SQLi detection tool identifies the parameterized URLs with the help of this detection mechanism procedure named "attack code generator". The attack code generator mechanism covers individual tasks by generating unique input data to solve these formulas. Afterwards it splits the parameterized URL and also regenerates the URL. Then it defines malicious payload and response patterns. To continue this process, it goes to the next step called "analyzer".

4.7 Analyzer

In the analyze segment have two phases- analyzer and compare. The analyzer phase starts its process by generating formulas. In compare phase- the formula will be executed to the combination of regenerated URL, payloads and firewall checker through user agent. The user agent collects responses for the remaining process.

4.8 Response Pattern

In this segment predefined response for SQL and Firewall will be defined. The main problem is that when the server is processed as a package, SQLi detection mechanism must have to decide whether the server will carry a specific hostile scenario. The detection mechanism will solve this complication by formulating a hostile script on how the responses on the host script server match the predefined response. After getting the result as SQLi vulnerability detected it ends up the procedure with storing the location.

4.9 FSM Construction

The float and mechanism of RCE model is built as a FSM (finite state machine) conceptually. In this area it will be described briefly.

An FSM is applied for mapping the flow and mechanism of RCE model as FSM is well matched for modeling the disposition of any model or system. The requisite records for creating FSM are accumulated from a testing environment, wherein it presents the parameters which have been used for establishing the FSM model.

FSM Construction

To tune out damaged authentication vulnerability, an annotated FSM is modeled with six tuples (Q, ∑, A, δ, q0,F)

Q denotes a finite set of states. In this model activity / vacation spot nodes are recognized as states.

∑ defines a finite set of inputs. The input symbols that make the transition to the subsequent state are illustrated as State which refer to the destination states in the FSM of the model.

A is the set of annotations. In this model they are represented as two tuples [P, S], which are being used to grant extra prerequisites to help the transition from one state to next state

P represents the set of HTTP parameters and their respective values which are passing along the http request.

S represents the set of Session Parameters described as session variables and similar to values.

δ is the transition function which is defined as Q x ∑ x A into Q.It might also be defined as, if δ(qs , i, a) = q1 it shows that if qs is the current nation with input symbol i (i ε ∑) below the annotation a(a ε A),then there will be transition from the present day kingdom qs to the subsequent state q1.This is called annotated FSM.

q0 Indicates the preliminary state, the place q0 ε Q.Herein Initial state is Input Url from the place the FSM starts.

F represents the last set of states, the place F ⊆ Q.Final state/states refer to the state/states that mark the quit state/states of a model.

The endeavor nodes are represented as states in the annotated FSM.The edges are labeled with Destination State , HTTP parameters and session variables that are input/set of inputs through the consumer or system.Whenever user/system enters a URL to generate request, it will be introduced to input symbol set(∑) and that will be checked with unique function HTTPConnection.The characteristic HTTPResponse will retrieve the HTTP parameters from the request and define the transition to the next nation with session variables as session parameter and Get parameters.The initial nation ‘Input URL’ will be blanketed in q0.The input symbols and annotation feature at a particular state ascertains transitions to a posterior state wherein,

[δ(particular state , enter symbol, annotation function) = Destination state]

RCE injection vulnerability provoked due to improper coding and design of RCE query which allows data to construct a RCE query dynamically. In the ADT-RCE model, the state "Seed URL" will be included in initial state q0. The input symbols and annotation function in a given state establishes transitions to a subsequent state and it is called transition function which is denoted by δ (particular state , input symbol, annotation function) = Destination state. The finite set of states Q = {Seed URL, Execute URL for Source Code Using BS4, Store Source Code in temp variable, Extract all links from source code, Select Parameterized URL, Select URLs Splitting the parameter, Regenerate URL, Formula = Regenerated URL+Payloads+firewall Checker, Execute

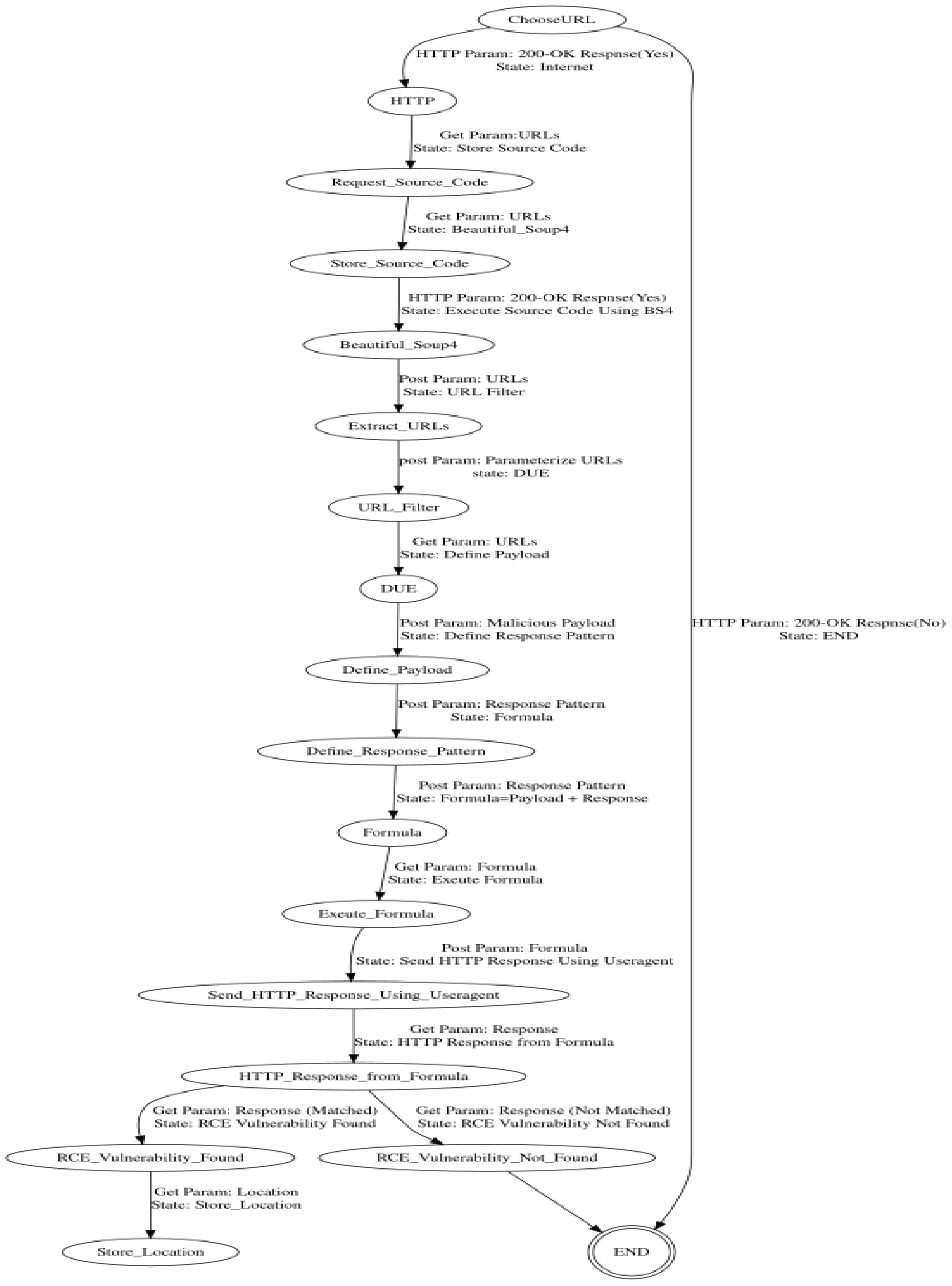

ChooseURL
HTTP Param: 200-OK Respnse(Yes) State: Internet
HTTP
Get Param:URLs State: Store Source Code
Request_Source_Code
Get Param: URLs State: Beautiful_Soup4
Store_Source_Code
HTTP Param: 200-OK Respnse(Yes) State: Execute Source Code Using BS4
Beautiful_Soup4
Post Param: URLs State: URL Filter
Extract_URLs
post Param: Parameterize URLs state: DUE
URL_Filter
Get Param: URLs State: Define Payload
DUE
Post Param: Malicious Payload State: Define Response Pattern
HTTP Param: 200-OK Respnse(No) State: END
Define_Payload
Post Param: Response Pattern State: Formula
Define_Response_Pattern
Post Param: Response Pattern State: Formula=Payload + Response
Formula
Get Param: Formula State: Execute Formula
Exeute_Formula
Post Param: Formula State: Send HTTP Response Using Useragent
Send_HTTP_Response_Using_Useragent
Get Param: Response State: HTTP Response from Formula
HTTP_Response_from_Formula
Get Param: Response (Matched) State: RCE Vulnerability Found
Get Param: Response (Not Matched) State: RCE Vulnerability Not Found
RCE_Vulnerability_Found
RCE_Vulnerability_Not_Found
Get Param: Location State: Store_Location
Store_Location
END

Figure 2: RCE Finite State Machine

Formula, Send HTTP Request Using User Agent, HTTP Response from Formula, RCE injection found, Store Location, RCE injection not found, END}. And the final states of ADT-RCE, F = {Store Location, End}

CHAPTER 5

IMPLEMENTATION AND VARIFICATION

A (partial) *Promela* version of this protocol is shown below. State S1 is the initial state in RCE model verification.

```
mtype = {live, offline, store_src_code,store_urls, execute,extract,
extract_urls, d_payload, d_response, c_response, generate,
check,match,notmatch,req_src_code,BS4,url_filter,g_formula }
chan Input=[0] of {mtype}
chan DataProcess=[0] of {mtype}
chan Crawler=[0] of {mtype}
chan AttackGenerator=[0] of {mtype}
chan UserAgent=[0] of {mtype}

active proctype Start()
{
S1:   Input!execute
S2:   if
      :: Input?live  -> goto S3
      :: Input?offline  -> goto S0
      fi
S3:   Crawler!req_src_code -> goto S4
S4:   Crawler!store_src_code -> goto S5
S5:   Crawler!BS4-> goto S6
S6:   Crawler!extract_urls  -> goto S7
S7:   Crawler!url_filter -> goto S8
S8:   Crawler!store_urls -> goto S9
S9:   AttackGenerator!d_payload -> goto S10
S10:  AttackGenerator!d_response -> goto S11
S11:  AttackGenerator!g_formula -> goto S12
S12:  UserAgent!c_response -> goto S13
S13:  if
      :: DataProcess?match -> goto S14
      ::DataProcess?notmatch -> goto S0
      fi
S14:  DataProcess!store_urls -> goto S15
S15:  printf("vulnerable URLs")
S0:
}
```

Hint: you can add fields to the messages with data values. To do so you have to change the channel declarations. For instance, as follows:

```
        chan Input=[0] of {mtype}
        chan DataProcess=[0] of {mtype}
        chan Crawler=[0] of {mtype}
        chan AttackGenerator=[0] of {mtype}
        chan UserAgent=[0] of {mtype}
```

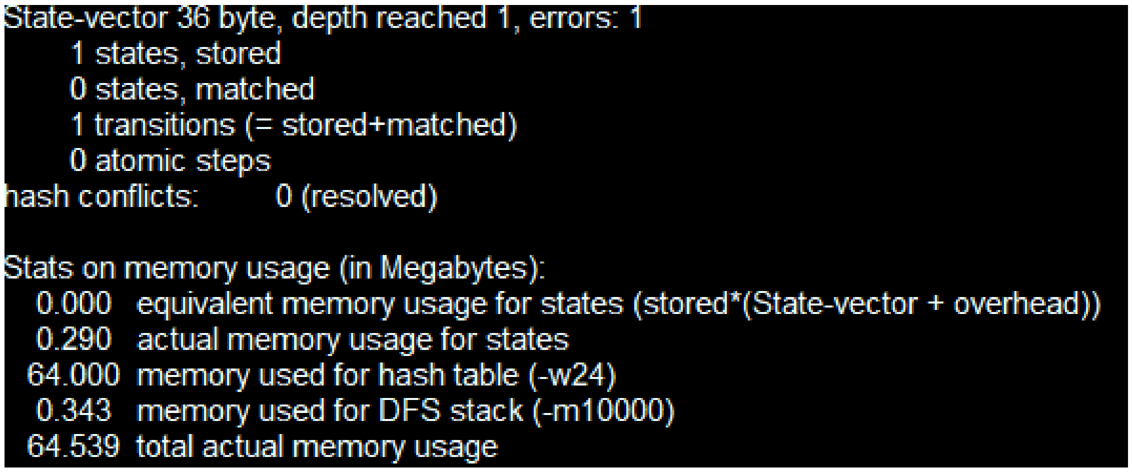
where the send and receive operations now have two field of matching types. For instance:

```
                a2b!a1(78)
```

5.1 VARIFICATION

For verifying the model use a spin verification to see if there are any unreachable states.

The verification result shown below.

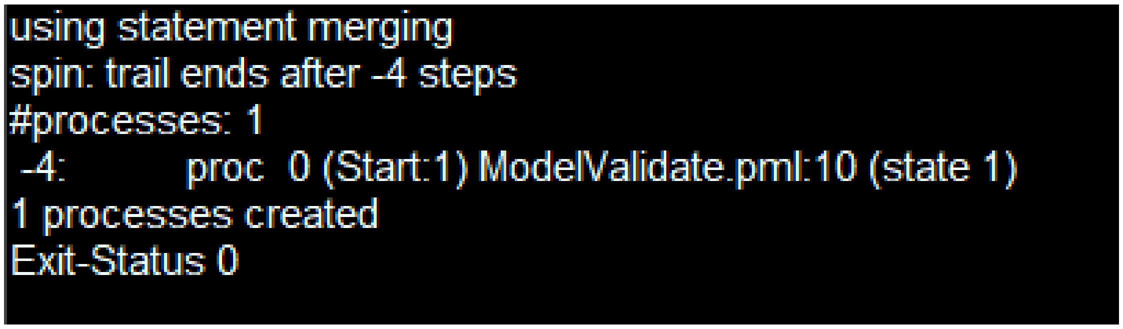


```
State-vector 36 byte, depth reached 1, errors: 1
       1 states, stored
       0 states, matched
       1 transitions (= stored+matched)
       0 atomic steps
hash conflicts:         0 (resolved)

Stats on memory usage (in Megabytes):
   0.000   equivalent memory usage for states (stored*(State-vector + overhead))
   0.290   actual memory usage for states
  64.000   memory used for hash table (-w24)
   0.343   memory used for DFS stack (-m10000)
  64.539   total actual memory usage
```

Figure 3: RCE Model Verification Screen Sort

We also simulate our model the simulation result shown below:



```
using statement merging
spin: trail ends after -4 steps
#processes: 1
 -4:          proc  0 (Start:1) ModelValidate.pml:10 (state 1)
1 processes created
Exit-Status 0
```

Figure 4: RCE Model simulation Result

The simulation clarifies that the model in finite state.

CHAPTER 6
CONCLUSION AND FUTURE RECOMMENDATION

6.1 Findings and Contributions

In these recent years, web application vulnerabilities have become a critical problem for all types of people who have been connected with the web. This research has presented an automated RCE vulnerability detection model, implemented a FSM based on the model, which is developed in windows platform. An examination has been performed on RCE FSM verification and the simulation result show the model in finite state. This study has observed that the biggest problem has been recognized as the insecure design of the applications and careless coding practice especially in using data/information retrieving methods.

6.2 Recommendations for Future Works

This research is a continuous process we will verify other web application vulnerability detection model using SPIN model checker.

REFERENCES

Begum, A., Hassan, M. M., Bhuiyan, T., & Sharif, M. H. (2016, December). RFI and SQLi based local file inclusion vulnerabilities in web applications of Bangladesh. In 2016 International Workshop on Computational Intelligence (IWCI) (pp. 21-25). IEEE.

Hassan, M. M., Ali, M. A., Bhuiyan, T., Sharif, M. H., & Biswas, S. Quantitative Assessment on Broken Access Control Vulnerability in Web Applications.

Yong Fang, Jiayi Peng, Liang Liu, and Cheng Huang. 2018. WOVSQLI: Detection of SQL Injection Behaviors Using Word Vector and LSTM. In Proceedings of the 2nd International Conference on Cryptography, Security and Privacy (ICCSP 2018). ACM, New York, NY, USA, 170-174. DOI: https://doi.org/10.1145/3199478.3199503

M. Liu and B. Wang, "A Web Second-Order Vulnerabilities Detection Method," in IEEE Access, vol.6,pp.70983-70988,2018. doi: 10.1109/ACCESS.2018.28810703.

Kevin Ross, Melody Moh, Teng-Sheng Moh and Jason Yao. "Multi-source data analysis and evaluation of machine learning techniques for SQL injection detection."March 2018 ACMSE '18: Proceedings of the ACMSE 2018 Conference. Doi: 10.1145/3190645.3190670

D. Kao, C. Lai and C. Su, "A Framework for SQL Injection Investigations: Detection, Investigation, and Forensics," 2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Miyazaki, Japan, 2018, pp. 2838-2843. doi: 10.1109/SMC.2018.00483

A. Algaith, P. Nunes, F. Jose, I. Gashi and M. Vieira, "Finding SQL Injection and Cross Site Scripting Vulnerabilities with Diverse Static Analysis Tools," 2018 14th European Dependable Computing Conference (EDCC), Iasi, 2018, pp. 57-64.

Nadar V.M., Chatterjee M., Jacob L. (2018) A Defensive Approach for CSRF and Broken Authentication and Session Management Attack. In: Perez G., Tiwari S., Trivedi M., Mishra K. (eds) Ambient Communications and Computer Systems. Advances in Intelligent Systems and Computing, vol 696. Springer, Singapore.

Lalia S., Sarah A. (2018) XSS Attack Detection Approach Based on Scripts Features Analysis. In: Rocha Á., Adeli H., Reis L., Costanzo S. (eds) Trends and Advances in Information Systems and Technologies. WorldCIST'18 2018. Advances in Intelligent Systems and Computing, vol 746. Springer, Cham. doi: https://doi.org/10.1007/978-3-319-77712-2_19

J. Pan and X. Mao, "Detecting DOM-Sourced Cross-Site Scripting in Browser Extensions," 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), Shanghai, 2017, pp. 24-34. doi: 10.1109/ICSME.2017.11.

Teodor Sommestad, Hannes Holm, Mathias Ekstedt, (2012) "Estimates of success rates of remote arbitrary code execution attacks", Information Management & Computer Security, Vol. 20 Issue: 2, pp.107-122,https://doi.org/10.1108/09685221211235625

T. Jain and N. Jain, "Framework for Web Application Vulnerability Discovery and Mitigation by Customizing Rules Through ModSecurity," 2019 6th International Conference on Signal Processing and Integrated Networks (SPIN), Noida, India, 2019, pp. 643-648. doi: 10.1109/SPIN.2019.8711673

B. Rexha, A. Halili, K. Rrmoku and D. Imeraj, "Impact of secure programming on web application vulnerabilities," 2015 IEEE International Conference on Computer Graphics, Vision and Information Security(CGVIS),Bhubaneswar,2015, pp. 61-66. doi: 10.1109/CGVIS.2015.7449894

Farah, T., Alam, D., Ali, M. N. B., & Kabir, M. A. (2015). Investigation of Bangladesh region based web applications: A case study of 64 based, local, and global SQLi vulnerability. 2015 IEEE International WIE Conference on Electrical and Computer Engineering (WIECONECE).doi:10.1109/wiecon-ece.2015.7443891

G. Deepa, P. Santhi Thilagam, Securing Web Applications from Injection and Logic Vulnerabilities: Approaches and Challenges, Information and Software Technology (2016), doi: 10.1016/j.infsof.2016.02.005

DivyaniYadav, D. Gupta, D. Singh, D. Kumar and U. Sharma, "Vulnerabilities and Security of Web Applications," 2018 4th International Conference on Computing Communication and Automation (ICCCA), Greater Noida, India, 2018, pp. 1-5. doi: 10.1109/CCAA.2018.8777558

S. Abaimov and G. Bianchi, "CODDLE: Code-Injection Detection With Deep Learning," in IEEE Access, vol. 7, pp. 128617-128627, 2019.doi: 10.1109/ACCESS.2019.2939870

L. Ma, D. Zhao, Y. Gao and C. Zhao, "Research on SQL Injection Attack and Prevention Technology Based on Web," 2019 International Conference on Computer Network, Electronic and Automation (ICCNEA), Xi'an, China, 2019, pp. 176-179.doi: 10.1109/ICCNEA.2019.00042

İ. KARA and M. AYDOS, "Detection and Analysis of Attacks Against Web Services by the SQL Injection Method," 2019 3rd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT), Ankara, Turkey, 2019, pp. 1-4. doi: 10.1109/ISMSIT.2019.8932755

T. Permpool, S. Nalintippayawong and K. Atchariyachanvanich, "Interactive SQL Learning Tool with Automated Grading using MySQL Sandbox," 2019 IEEE 6th International Conference on Industrial Engineering and Applications (ICIEA), Tokyo, Japan, 2019, pp. 928-932. doi: 10.1109/IEA.2019.8715175

Q. Li, W. Li, J. Wang and M. Cheng, "A SQL Injection Detection Method Based on Adaptive Deep Forest," in IEEE Access, vol. 7, pp. 145385-145394, 2019. doi: 10.1109/ACCESS.2019.2944951

L. Zhang, H. Zhang, X. Zhang and L. Chen, "A New Mechanism for Trusted Code Remote Execution," 2007 International Conference on Computational Intelligence and Security Workshops (CISW 2007), Heilongjiang, 2007, pp. 574-578. doi: 10.1109/CISW.2007.4425561

J. Wu, A. Arrott and F. C. C. Osorio, "Protection against remote code execution exploits of popular applications in Windows," 2014 9th International Conference on Malicious and Unwanted Software: The Americas (MALWARE), Fajardo, PR, 2014, pp. 26-31. doi: 10.1109/MALWARE.2014.6999416

Q. H. Mahmoud, D. Kauling and S. Zanin, "Hidden android permissions: Remote code execution and shell access using a live wallpaper," 2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, 2017, pp. 599-600. doi: 10.1109/CCNC.2017.7983184

M. Carlisle and B. Fagin, "IRONSIDES: DNS with no single-packet denial of service or remote code execution vulnerabilities," 2012 IEEE Global Communications Conference (GLOBECOM), Anaheim, CA, 2012, pp. 839-844. doi: 10.1109/GLOCOM.2012.6503217

Alsmadi, I., & Mira, F. (2018, April). Website security analysis: variation of detection methods and decisions. In 2018 21st Saudi Computer Society National Computer Conference (NCC) (pp. 1-5). IEEE.

Alwan, Z. S., & Younis, M. F. (2017). Detection and prevention of SQL injection attack: A survey. International Journal of Computer Science and Mobile Computing, 6(8), 5-17.

Dalai, A. K., & Jena, S. K. (2017). Neutralizing SQL injection attack using server side code modification in web applications. Security and Communication Networks, 2017.

Deepa, G., Thilagam, P. S., Praseed, A., & Pais, A. R. (2018). DetLogic: A black-box approach for detecting logic vulnerabilities in web applications. Journal of Network and Computer Applications, 109, 89-109.

Lawal, M. A., Sultan, A. B. M., & Shakiru, A. O. (2016). Systematic literature review on SQL injection attack. International Journal of Soft Computing, 11(1), 26-35.

Hassan, M. M., Bhuyian, T., Sohel, M. K., Sharif, M. H., & Biswas, S. (2018). SAISAN: An automated Local File Inclusion vulnerability detection model. International Journal of Engineering & Technology, 7(2-3), 4.

Hassan, M. M., Nipa, S. S., Akter, M., Haque, R., Deepa, F. N., Rahman, M., ... & Sharif, M. H. (2018). Broken authentication and session management vulnerability: A case study of Web application. International Journal of Simulation Systems, Science & Technology, 19(2), 6-1.

Code Injection. (n.d.). Retrieved from https://owasp.org/www-community/attacks/Code_Injection